Prezentacja projektu
mPlayer
Programowanie III

Dawid Bitner
Marcin Krupa

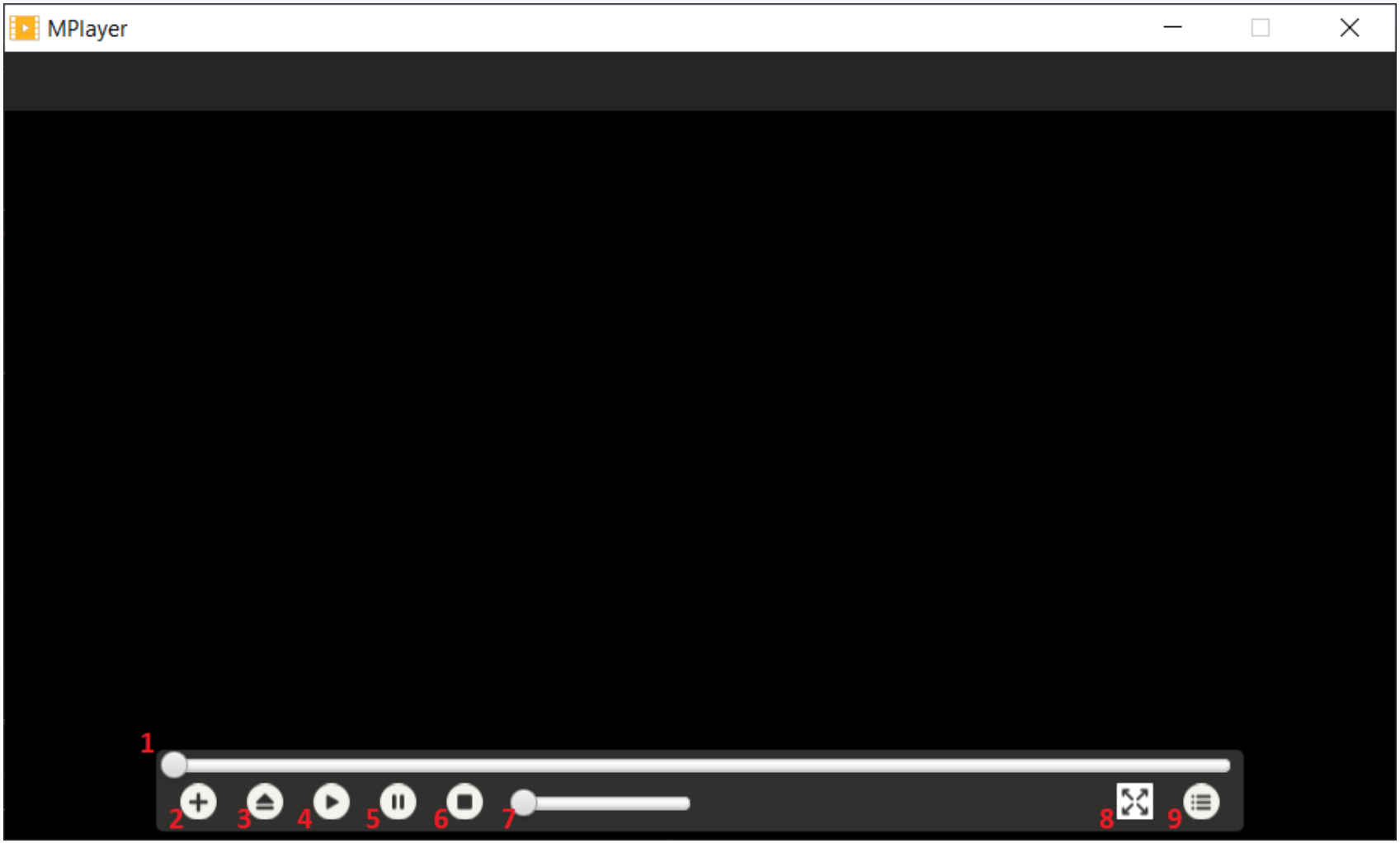
Politechnika Śląska
Matematyka Stosowana

Spis treści

Opis programu

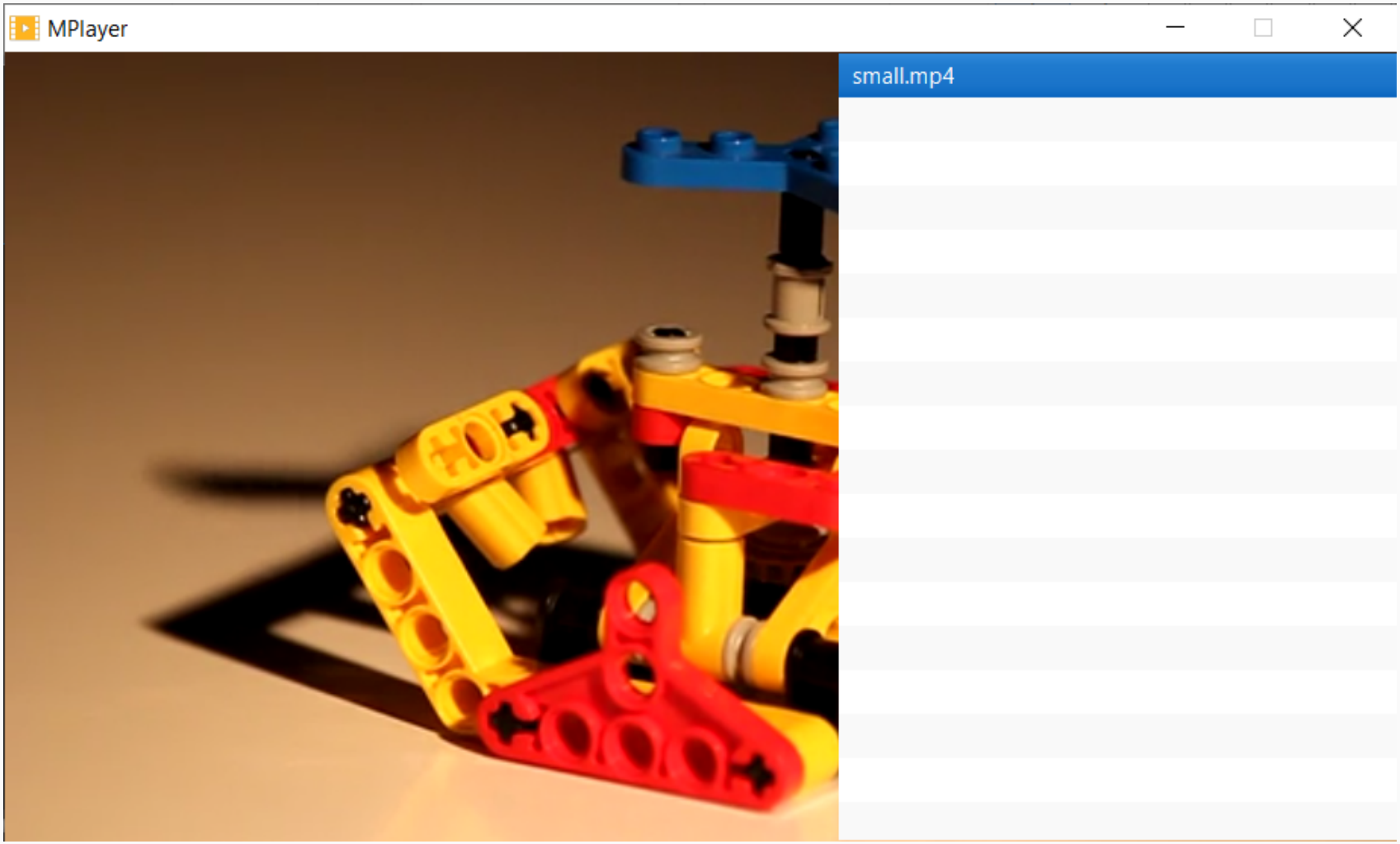
Celem programu jest umożliwienie użytkownikowi odtworzenie wybranych przez niego plików wideo zapisanych w formacie *.mp4*.

Instrukcja obsługi



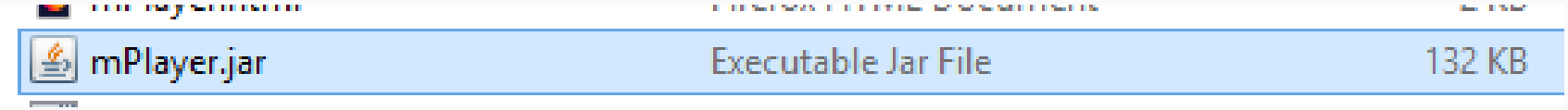
[1] suwak postępu
[2] dodanie pliku do playlisty
[3] dodanie zawartości folderu do playlisty
[4] odtwarzanie
[5] pauza
[6] zatrzymanie odtwarzania
[7] suwak poziomu głośności
[8] przełączanie pomiędzy trybem pełnoekranowym
[9] lista odtwarzania

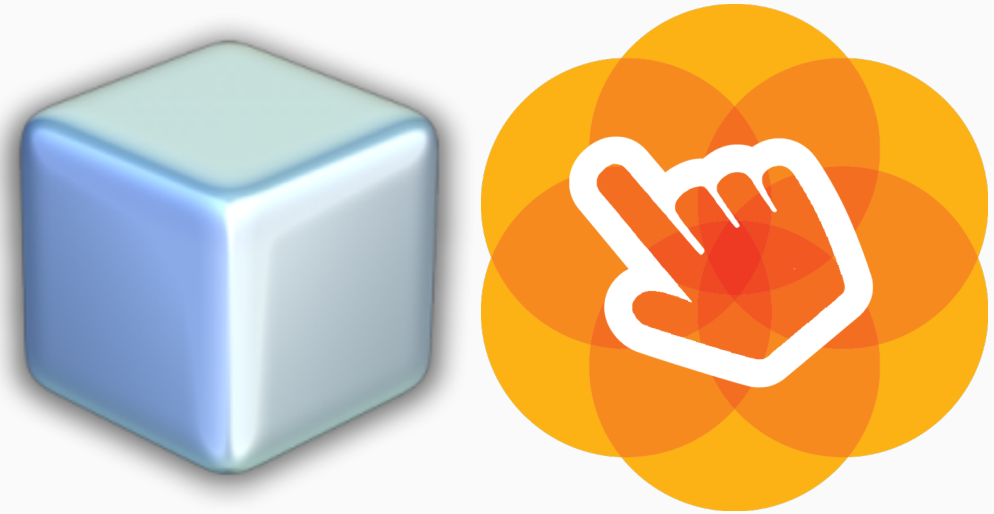
Odtwarzacz z widocznym wysuniętym menu zawierającym listę odtwarzania:



Część techniczna

Program uruchamiamy poprzez wywołanie pliku *mPlayer.jar*. W celu zapewnienia poprawnego działania programu potrzebujemy systemu operacyjnego z zainstalowanym środowiskiem *Java* w wersji *8* lub nowszej. Program został napisany przy pomocy zintegrowanego środowiska programistycznego *NetBeans IDE 8.2*, strona wizualna projektu została stworzona przy pomocy *JavaFX Scene Builder 8.5.0*, darmowego narzędzia do edycji plików technologii *JavaFX* firmy *Gluon*, na której to został oparty projekt.





Podział projektu

1. MPlayer.java
2. FXMLDocument.fxml
3. FXMLDocumentController.java

MPlayer.java

Plik ten zawiera główną klasę programu o nazwie *MPlayer*, dziedziczącą po wbudowanej klasie *Application* z rodziny *JavaFX*. Znajdują się tutaj główne metody tj. *main()* i *start()*. Plik zawiera obiekt:

```
AnchorPane root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
```

Odpowiada on za pobranie hierarchii obiektów z pliku XML, które są obiektami graficznymi uwzględnionymi w projekcie rozszerzając jego funkcjonalność. Obiekt *root* jest obiektem klasy *AnchorPane*, jest zatem głównym panelem projektu.

```
Scene scene = new Scene(root);
```

Obiekt *root* służy do zainicjalizowania obiektu klasy *Scene*, który jest kontenerem dla całej zawartości sceny projektu, który jest wyświetlany użytkownikowi. O ile *root* zawiera bardziej strukturę logiczną projektu, o tyle obiekt *scene* ma tę strukturę wyświetlić użytkownikowi.

```
FXMLDocumentController controller = new FXMLDocumentController();
```

Kolejnym istotnym obiektem jest obiekt *controller* klasy *FXMLDocumentController*. Inicjalizuje on wszelkie metody sterujące aplikacją zawarte w pliku *FXMLDocumentController.java*.

Następnie, poprzez wywołanie odpowiednich metody klasy *Stage* dla obiektu *stage*, będącego nadrzędnym kontenerem całego projektu zostały określone takie właściwości jak startowy rozmiar okna, jego nazwa, ikona programu, czy to ja ma się zachowywać gdy zostanie wychwycone podwójne naciśnięcie lewego przyciska myszy.

```
stage.setTitle("MPlayer");
stage.getIcons().add(new Image(getClass().getResourceAsStream("/images/video-player.png")));
stage.setResizable(false);

scene.setOnMouseClicked((MouseEvent doubleClicked) -> {
    if(doubleClicked.getButton().equals(MouseButton.PRIMARY)){
        if(doubleClicked.getClickCount() == 2){
            if (stage.isFullScreen() == false) {
                stage.setFullScreenExitHint("");
                stage.setFullScreen(true);
            }
            else stage.setFullScreen(false);
        }
    }
});

stage.setScene(scene);
stage.show();
```

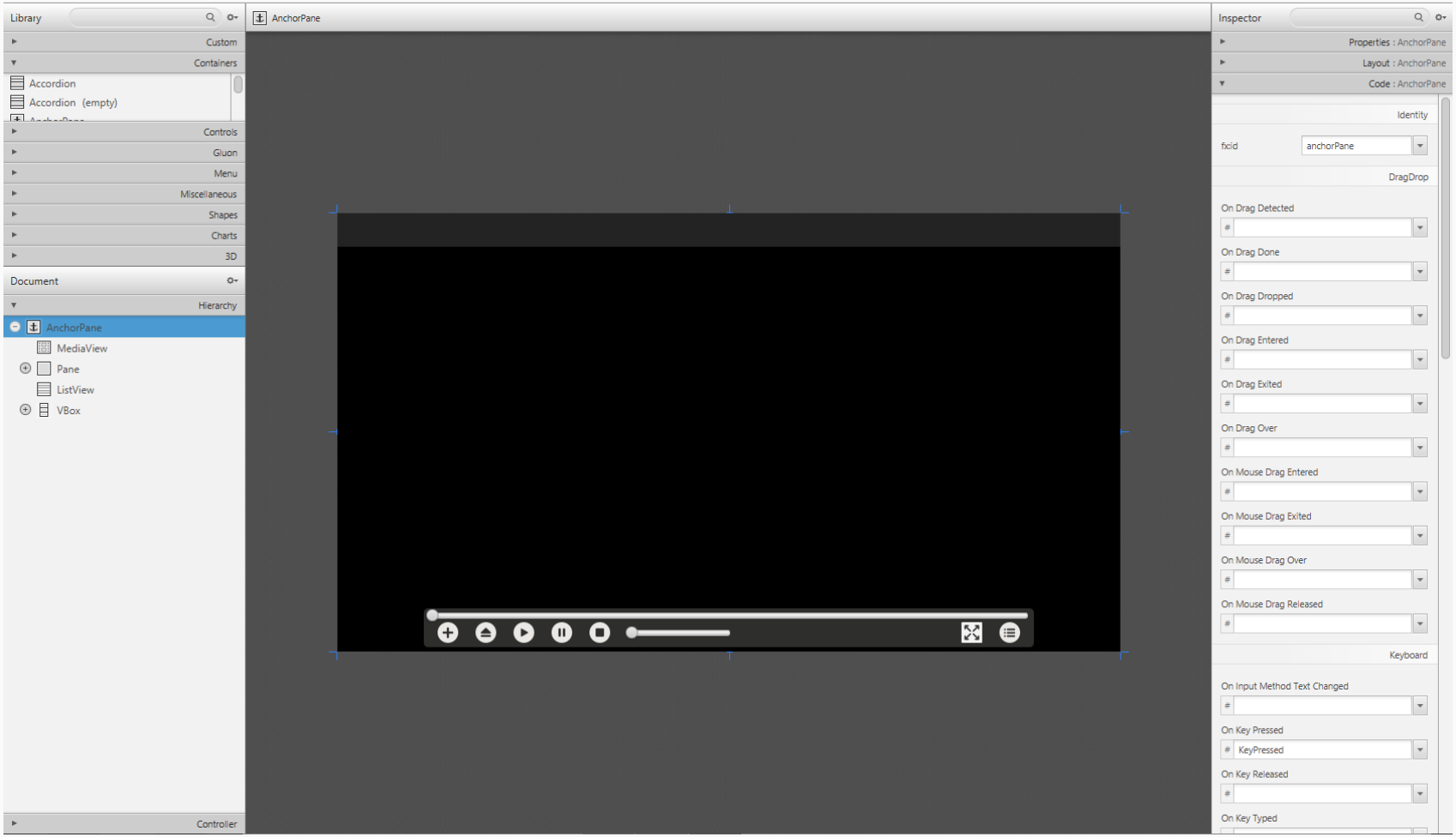
FXMLDocument.fxml

Plik wygenerowany na podstawie zmian dokonywanych za pomocą *JavaFX Scene Builder* na strukturze obiektów XML. Plik jest oparty o *XML/FXML*. Zawiera on informacje o obiektach umieszczonych w projekcie, ich hierarchii oraz edycji obiektów w kwestiach takich jak zmiana układu, prywatnych właściwości obiektów, czy kodu (przypisanie ID czy wywołanie odpowiednich metod typu Action Methods).

Część kodu pliku *FXMLDocument.fxml*

```
<AnchorPane fx:id="anchorPane" onKeyPressed="#KeyPressed" prefHeight="508.0" prefWidth="907.0" stylesheets=
"@style.css" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller=
"mplayer.FXMLDocumentController">
   <children>
      <MediaView fx:id="mediaView" nodeOrientation="INHERIT" />
      <Pane fx:id="descriptionPanel" onMouseEntered="#PaneOnMouseEntered" onMouseExited="#PaneOnMouseExited"
       prefHeight="39.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
         <children>
            <Label fx:id="currentlyPlaying" layoutX="22.0" layoutY="10.0" onMouseEntered=
            "#PaneOnMouseEntered" onMouseExited="#PaneOnMouseExited" prefHeight="16.0" prefWidth="460.0"
            textFill="WHITE">
               <font>
                  <Font size="14.0" />
               </font></Label>
         </children>
      </Pane>
      <ListView fx:id="playlist" fixedCellSize="0.0" nodeOrientation="LEFT_TO_RIGHT" onMouseClicked=
      "#selectItem" prefWidth="370.0" AnchorPane.bottomAnchor="0.0" AnchorPane.rightAnchor="0.0"
      AnchorPane.topAnchor="0.0" />
      <VBox fx:id="controlBar" alignment="CENTER" layoutX="133.0" layoutY="405.0" onMouseEntered=
      "#cBarOnMouseEntered" onMouseExited="#cBarOnMouseExited" prefHeight="45.0" prefWidth="497.0"
      AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="100.0" AnchorPane.rightAnchor="100.0">
         <children>
            <Slider fx:id="progSlider" majorTickUnit="5.0" max="200.0" prefHeight="14.0" prefWidth="462.0" />
            <HBox alignment="CENTER" prefHeight="31.0" prefWidth="689.0">
               <children>
                  <Button fx:id="openFileButton" minWidth="40.0" mnemonicParsing="false" onAction=
```

Edycja pliku *FXMLDocument.fxml* aplikacją *SceneBuilder*



FXMLDocumentController.java

Plik ten zawiera najważniejsze metody sterujące aplikacją. Wsród nich:

- checkResolution()
- addSingleFile()
- addFolder()
- selectItem()
- fullScreen()
- openListView()
- cBarOnMouseEntered() oraz cBarOnMouseExited()
- PaneOnMouseEntered() oraz PaneOnMouseExited()
- KeyPressed()
- playFile()

checkResolution()

Funkcja ta ma za zadanie określenie rozdzielczości ekranu użytkownika. Korzysta ona z klasycznego rozwiązania którym jest obiekt klasy *Dimensions*, pobierający rozdzielczość z klasy *Toolkit* z funkcji *getDefaultToolkit* oraz *getScreenSize*. Na podstawie uzyskanych danych, program jest w stanie dostować rozmiar okna do rodzielczości stosowanej przez użytkownika.

```
@FXML
protected boolean checkResolution () {
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    double width = screenSize.getWidth();
    double height = screenSize.getHeight();

    return (width/height)<1.7;
}
```

addSingleFile()

Funkcja dodaje jeden wybrany plik przez użytkownika do playlisty. Wykorzystuje ona klasę FileChooser, oraz jej metody jak *ExtensionFilter* do filtrowania rozszerzeń plików czy *showOpenDialog* do wyświetlenia okienka eksploratora do wyboru pliku.

```
@FXML
private void addSingleFile(ActionEvent event) {

    FileChooser fileChooser = new FileChooser();
    FileChooser.ExtensionFilter filter = new FileChooser.ExtensionFilter("Select mp4 file", "*.mp4");

    fileChooser.getExtensionFilters().add(filter);

    File file = fileChooser.showOpenDialog(null);

    playlist.setCellFactory(TextFieldListCell.forListView());
    playlist.getItems().add(index, file.getName());
    filesURI.add(file.toURI().toString());
    index++;
}
```

addFolder()

Korzysta z klasy DirectoryChooser. Wybrany folder, a dokładniej pliki z odpowiednim rozszerzeniem, przekazywane są do listy która jest następnie przekazywana do obiektu *playlist* klasy *ListView*, będącym menu wyboru załadowanych plików.

```
@FXML
private void addFolder(ActionEvent event) {
    DirectoryChooser directoryChooser = new DirectoryChooser();
    File folder = directoryChooser.showDialog(null);

    //filter na rozszerzenia plików bierze pod uwagę tylko .mp3 i .mp4
    listOfFiles = folder.listFiles((File folder1, String name) -> {
        return name.toLowerCase().endsWith(".mp4");
    } );

    playlist.setCellFactory(TextFieldListCell.forListView());

    for (File listOfFile : listOfFiles) {
        if (listOfFile.isFile()) {
            playlist.getItems().add(index, listOfFile.getName());
            filesURI.add(listOfFile.toURI().toString());
            index++;
        }
    }
}
```

selectItem()

Funkcja pozwala wybrać interesujący użytkownika plik wideo poprzez dwuklik. Nazwa pliku jest wyświetlana także w panelu na górze aplikacji.

```
@FXML
public void selectItem(MouseEvent click) {
    if (click.getClickCount() == 2) {

        int currentItemSelected = playlist.getSelectionModel().getSelectedIndex();

        currentlyPlaying.setText(playlist.getSelectionModel().getSelectedItem());
        playFile(filesURI.get(currentItemSelected));
    }
}
```

fullScreen()

Pozwala zmienić tryb na pełnoekranowy z wykorzystaniem przycisku z punktu [8] opisu obsługi programu. Tryb pełnoekranowy można także wywołać dwuklikiem.

```
@FXML
private void fullScreen(ActionEvent event) {
    Stage stage = (Stage) fsButton.getScene().getWindow();
    if (stage.isFullScreen()) stage.setFullScreen(false);
    else {
        stage.setFullScreenExitHint("");
        stage.setFullScreen(true);
    }

}
```

openListView()

Otwiera playlistę z prawej strony aplikacji.

```
@FXML
private void openListView (ActionEvent event) {
    if (playlist.isVisible()) playlist.setVisible(false);
        else playlist.setVisible(true);
}
```

cBarOnMouseEntered() oraz cBarOnMouseExited()

Control Bar aplikacji, czyli panel odpowiedzialny za wszystkie przyciski sterujące aplikacją. Reaguje na najechanie myszką przez użytkownika.

```
@FXML
private void cBarOnMouseEntered (MouseEvent event) {
    if (controlBar.getOpacity()==0) {
        FadeTransition ft = new FadeTransition(Duration.millis(150), controlBar);
        ft.setFromValue(0);
        ft.setToValue(1);
        ft.play();
    }
}

@FXML
private void cBarOnMouseExited (MouseEvent event) {
    FadeTransition ft = new FadeTransition(Duration.millis(150), controlBar);
    ft.setFromValue(1);
    ft.setToValue(0);
    ft.play();
}
```

PaneOnMouseEntered() oraz PaneOnMouseExited()

Panel na górze aplikacji. Reaguje na najechanie myszką przez użytkownika. Wyświetla aktualnie grany plik wideo.

```
@FXML
private void PaneOnMouseEntered (MouseEvent event) {
    if (descriptionPanel.getOpacity()==0) {
        FadeTransition ft = new FadeTransition(Duration.millis(150), descriptionPanel);
        ft.setFromValue(0);
        ft.setToValue(1);
        ft.play();
    }
}

@FXML
private void PaneOnMouseExited (MouseEvent event) {
    FadeTransition ft = new FadeTransition(Duration.millis(150), descriptionPanel);
    ft.setFromValue(1);
    ft.setToValue(0);
    ft.play();
}
```

KeyPressed()

Funkcja która odpowiada za zatrzymywanie/odtwarzanie pliku wideo po naciśnięciu spacji przez użytkownika.

```
@FXML
private void KeyPressed (KeyEvent event) throws IOException {
    if(event.getCode() == KeyCode.SPACE){
        if (isPlaying == false) {
            mediaPlayer.play();
            isPlaying = true;
        } else {
            mediaPlayer.pause();
            isPlaying = false;
        }
    }
}
```

playFile()

Funkcja główna odpowiedzialna za odtwarzanie pliku wideo. Korzysta ona z obiektów: *media*, *mediaPlayer* oraz *mediaView* odpowiednio klas *Media*, *MediaPlayer* oraz *MediaView*. Jednym z jej zadań, poza odtwarzaniem pliku, jest kontrola wielkości okna aplikacji oraz: zmian w pasku progresu odtwarzania, aktualizacji czasu odtwarzania pliku, zmiany głośności odtwarzania poprzez wykorzystanie słuchaczy, tj. interfejsów klasy *ActionListener*. Funkcja ta także odpowiada za zmiany w progresie odtwarzania, gdy użytkownik kliknie w interesujący go fragment na pasku odtwarzania wideo przeskoczy do tego momentu.

```
@FXML
private void playFile(String file) {
    playlist.setVisible(false);
    if (isPlaying == true) mediaPlayer.dispose();
    isPlaying=true;
    Media media = new Media(file);
    mediaPlayer = new MediaPlayer(media);
    mediaView.setMediaPlayer(mediaPlayer);
    if (checkResolution()) mediaView.setY(50.0);
    mediaPlayer.play();

    DoubleProperty width = mediaView.fitWidthProperty();
    DoubleProperty hight = mediaView.fitHeightProperty();

    width.bind(Bindings.selectDouble(mediaView.sceneProperty(),"width"));
    hight.bind(Bindings.selectDouble(mediaView.sceneProperty(),"hight"));

    progSlider.valueProperty().addListener((observable, oldValue, newValue) -> {
        progSlider.setMax(mediaPlayer.getTotalDuration().toMillis());
    });

    volumeSlider.setValue(mediaPlayer.getVolume()*100);

    volumeSlider.valueProperty().addListener((Observable observable) -> {
        mediaPlayer.setVolume(volumeSlider.getValue()/100);
    });

    mediaPlayer.currentTimeProperty().addListener((ObservableValue<? extends Duration> observable,
    Duration oldValue, Duration newValue) -> {
        progSlider.setValue(newValue.toMillis());
    });
```

```
progSlider.setOnMousePressed((MouseEvent event1) -> {
    mediaPlayer.pause();
});

progSlider.setOnMouseClicked((MouseEvent event1) -> {
    mediaPlayer.seek(Duration.millis(progSlider.getValue()));
    mediaPlayer.play();
});

progSlider.setOnMouseReleased((MouseEvent event1) -> {
    mediaPlayer.seek(Duration.millis(progSlider.getValue()));
    mediaPlayer.play();
});

mediaPlayer.currentTimeProperty().addListener((observable) -> {
    hrsFull = String.valueOf(((int) mediaPlayer.getTotalDuration().toHours()));
    if (Integer.valueOf(hrsFull)  < 10) hrsFull="0"+hrsFull;
    minFull = String.valueOf(((int) mediaPlayer.getTotalDuration().toMinutes())%60);
    if (Integer.valueOf(minFull)  < 10) minFull="0"+minFull;
    secFull = String.valueOf(((int) mediaPlayer.getTotalDuration().toSeconds())%60);
    if (Integer.valueOf(secFull)  < 10) secFull="0"+secFull;
    fullTime = String.format("%s:%s:%s", hrsFull, minFull, secFull);
});

mediaPlayer.currentTimeProperty().addListener((Observable observable) -> {
    hrs = String.valueOf(((int) mediaPlayer.getCurrentTime().toHours()));
    if (Integer.valueOf(hrs) < 10) hrs="0"+hrs;
    min = String.valueOf(((int) mediaPlayer.getCurrentTime().toMinutes())%60);
    if (Integer.valueOf(min) < 10) min="0"+min;
    sec = String.valueOf(((int) mediaPlayer.getCurrentTime().toSeconds())%60);
    if (Integer.valueOf(sec) < 10) sec="0"+sec;

    displayTime.setText(hrs+":"+min+":"+sec+"/"+fullTime);
```

style.css

Zawiera zmiany szaty graficznej projektu. Korzysta z technologii *Cascading Style Sheets*.

```
.button{
    -fx-background-color: transparent;
}

.button:hover {
    -fx-background-color: rgba(170, 170, 112, .5);
}

#playlist {
    -fx-background-color: transparent;
}

.list-cell:filled:selected:focused, .list-cell:filled:selected {
    -fx-background-color: linear-gradient(#328BDB 0%, #207BCF 25%, #1973C9 75%, #0A65BF 100%);
    -fx-text-fill: white;
}

.list-cell:even {
    -fx-background-color: white;
}

.list-cell:filled:hover {
    -fx-background-color: #0093ff;
    -fx-text-fill: white;
}

#descriptionPanel {
    -fx-background-color: rgba(75, 75, 74, .5);
}

#controlBar {
```

Użyte biblioteki

```
import java.awt.Dimension;
import java.awt.Toolkit;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.*;
import javafx.animation.FadeTransition;
import javafx.beans.Observable;
import javafx.beans.binding.Bindings;
import javafx.beans.property.DoubleProperty;
import javafx.beans.value.ObservableValue;
```

Użyte biblioteki

```
import javafx.event.ActionEvent;
import javafx.fxml.*;
import javafx.scene.Parent;
import javafx.scene.control.*;
import javafx.scene.control.cell.TextFieldListCell;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.media.*;
import javafx.stage.*;
import javafx.util.Duration;
import javafx.scene.layout.*;
```

Dziękujemy za uwagę